VZ200 VIDEO HARDWARE INTERRUPT

Steve Olney

This article details how to use the video hardware interrupt on the VZ200 and gives three simple examples of its usefulness.

THE HARDWARE INTERRUPT is a very useful feature of a computer's capability, with many different applications. The usefulness comes from the ability to 'interrupt' the normal flow of software execution, diverting the operation of the CPU by external means. The CPU can then be made to execute a separate, independent program before returning to the original program execution.

This description may sound like a GOSUB call to a subroutine in Basic, or a CALL to a subroutine in a machine code program, but there is an important difference. The difference is that the interrupt can occur asynchronously to the normal program execution (that is, it can occur at any time unrelated to the progress of normal program execution).

This capability is extremely useful when the computer has to serve some external device which can't wait for an action by the computer during normal program execution. Such devices range from a digital-to-analogue converter (which must sample data at strictly regular intervals), to a software clock counter which needs to be incremented by an external hardware clock pulse. By using a hardware interrupt these devices can be served almost immediately, in the time it takes the CPU to complete the current instruction.

The interrupt is called a hardware interrupt because there is a special pin on the CPU chip itself, which, when taken to ground potential (low or zero), initiates the interrupt sequence. This action is also performed by some external hardware device.

The VZ200 uses a Z80 CPU chip, which has three different responses to this interrupt signal depending on the interrupt mode set in the internal interrupt register (IR). Note that we are talking about the $\overline{\text{INT}}$ case, not the $\overline{\text{NMI}}$). For the VZ200 the interrupt register is set to interrupt mode 1 (by an IM1 instruction) during the initialization sequence.

The response to an interrupt in Interrupt Mode 1 is to complete the current instruction, save the program counter register (PCR) contents on the stack (allowing resumption of execution at that point upon returning from the interrupt) and then jump to location 0038 HEX. This could be viewed as a hardware version of the software RST 38 instruction.

The VZ200 video interrupt

Those of you who have access to a circuit diagram of the VZ200 will see that the interrupt pin (pin 16 $\overline{\text{INT}}$) of the Z80 CPU is connected to pin 37 ($\overline{\text{FS}}$) of the 6847 video controller chip. Reference to the 6847 data sheets shows that pin 37 of the 6847 chip is the video field sync output pin. This pin is pulled low by the 6847 chip during the vertical retrace period of the video output signal. That is, the field sync output pin goes low every 1/50 of a second (video frame rate of 50 per second) causing the Z80 CPU to be interrupted and diverted to location 0038 HEX every 20 ms.

Scrutiny of the machine code (in ROM) at location 0038 HEX reveals a JUMP instruction to location 2EB8 HEX. This jump is referred to as interrupt vector.

The machine code at 2EB8 HEX contains several CALLs to various locations before returning to the original program execution. I haven't looked at these in detail, but most likely they are concerned with cursor control and perhaps screen scrolling during listing.

In any case, the code in which we are interested is near the start of the code at 2EB8 HEX. The first CALL after saving affected registers is to location 787D HEX. There are two interesting points to note here. The first is that location 787D HEX is in RAM, and secondly, this is the memory location referred to in the VZ200 Technical Manual (under System pointers) as the "interrupt exit".

By PEEKing location 787D HEX (eg ▶

LISTING 1

```
HEX CODE        MNEMONIC
F5              PUSH AF          ; Save 'AF' register because we alter it
3E 2A           LD   A,2AH       ; Load 'A' register with code for '*'
32 1F 70        LD   (701FH),A   ; Put it in the top right-hand corner of screen
F1              POP AF           ; Restore 'AF' register
C9              RET              ; Return
```

LISTING 2

```
100  S= -32768 : F = S + 7 :' START AT 8000 HEX
200  FOR I = S TO F        :' POKE THE 8-BYTE MACHINE CODE PROGRAM
300   READ D               :' INTO MEMORY STARTING AT 8000 HEX
400   POKE I,D             :'
500  NEXT I                :'
600  POKE 30846,00         :' ENTER THE START ADDRESS OF THE MACHINE
700  POKE 30847,128        :' CODE PROGRAM INTO INTERRUPT JUMP
800  POKE 30845,195        :' EXIT AT 787D HEX.
900  DATA 245,62,42,50,31,112,241,201:'  DECIMAL EQUIVALENT OF HEX
```

LISTING 3

```
HEX CODE        MNEMONIC
F5              PUSH AF          ; save registers
C5              PUSH BC          ; we destroy
E5              PUSH HL          ;
3A 3B 78        LD   A,(783BH)   ; load latch contents
06 08           LD   B,8         ; bit counter
21 18 70        LD   HL,7018H    ; start of screen display
17        LOOP  RLA              ; rotate into carry and test
30 07           JR   NC,ZERO     ;
36 31           LD   (HL),31H    ; output '1'
23              INC  HL          ; adjust to next display position
10 F8           DJNZ LOOP        ; go until all bits are done
18 05           JR   EXIT        ; exit if done
36 30     ZERO  LD   (HL),30H    ; output '0'
23              INC  HL          ; adjust to next screen position
10 F1           DJNZ LOOP        ; go until all bits are done
E1        EXIT  POP  HL          ; exit
C1              POP BC           ;
F1              POP AF           ;
C9              RETURN           ;
```

LISTING 4

```
100  S= -32768 : F = S + 29 : 'START AT 8000 HEX
200  FOR I = S TO F          :' POKE THE 8-BYTE MACHINE CODE PROGRAM
300   READ D                 :' INTO MEMORY STARTING AT 8000 HEX
400   POKE I,D               :'
500  NEXT I                  :'
600  POKE 30846,00           :' ENTER THE START ADDRESS OF THE MACHINE
700  POKE 30847,128          :' CODE PROGRAM INTO INTERRUPT JUMP
800  POKE 30845,195          :' EXIT AT 787D HEX.
900  DATA 245,197,229,58,59,120,6,8
1000 DATA 33,24,112,23,48,7,54,49
1100 DATA 35,16,248,24,5,54,48,35
1200 DATA 16,241,225,193,241,201
```

PRINT PEEK[30845]) you should find it contains 201 DECIMAL (0C9 HEX) which is the Z80 RETurn instruction.

Using the video interrupt

Let's just back up to summarize what we've discussed so far. Every 20 ms the Z80 CPU is interrupted by the 6847 video controller chip. The interrupt mode (mode 1) causes the Z80 to jump to location 0038 HEX. From here execution jumps to 2EB8 HEX where a CALL to 787D HEX is encountered. Location 787D HEX (in RAM) contains a RET instruction and so execution returns immediately and continues until 2EDA HEX where a return from interrupt instruction (RETI) is found. Execution is now RETurned to the original program flow.

Now, because location 787D HEX is in RAM, we can change the RET instruction at that location to a JUMP to some other selected location. At this location we can insert our own interrupt servicing code.

Here is a very simple example to illustrate this procedure. Starting at location 3450 HEX in the Basic ROM is a subroutine which generates the 'beep' whenever you press a key. We can alter location 787D, 787E and 787F HEX to contain a JUMP to 3450 HEX to execute this 'beep' routine every time a video interrupt occurs (every 20 ms).

To do this we POKE the following machine code into memory starting at location 787D HEX:

Hex Code	Mnemonic
C3 50 34	JP 3450H

Note: Remember location 787D HEX is CALLed every 20 ms, so you must not alter the RET at this location until you have entered a valid jump address in the following two bytes. Otherwise the Z80 will jump to some indeterminate address depending on what random data was contained in 787E and 787F HEX.

The following strict order should be used:

POKE 30846,80 (POKE 50 HEX into location 787E HEX)
POKE 30847,52 (POKE 34 HEX into location 787F HEX)
POKE 30845,195 (POKE C3 HEX into location 787D HEX)

Type in the above commands via the immediate mode (without line numbers). The text within the brackets should *not* be typed in as it is for information only.

Once you have done this you should hear an almost continuous beep from the internal speaker. Notice that there is nothing which interferes with this beeping. Well, almost nothing, as will be explained a little later. However, you can enter a Basic program as normal (except for the distraction of the beeping) and even RUN or LIST it. In fact, you can do all the normal operations (ex-

cept tape operations — see below) without affecting the beeping. This is because the interrupt has priority over other software execution. So we see it is possible to have a Basic program running in the 'foreground' with a separate machine language program running in the 'background' being executed at regular intervals.

To stop the beep all that is necessary is to change the JUMP instruction (0C3 HEX) at location 787D HEX back to a RET (0C9 HEX) by:

POKE 30845,201

Tape operations

As mentioned earlier, there is another action which will disable the 'beep'. During tape operations, interrupts are disabled to ensure that accurate timing delays in the tape function's machine code are not disturbed. So while you are CSAVEing, CRUNning or CLOADing data to or from tape the beeping will stop. However, once the operation is over the interrupts are enabled once again and the beeps return.

To enable the 'beep' again, enter —

POKE 30845,195

Note: Before typing the above, make sure that locations 787E and 787F HEX contain the correct jump address (3450 HEX)!

Non erasable video display

Next we'll look at an example which shows how the video interrupt can be used to put 'non-erasable' information on the video screen.

Normally, any information displayed on the screen can be overwritten, cleared or scrolled off the screen, either during program execution or in the immediate execution mode. By using the video interrupt you can display information which cannot be overwritten.

The machine language source code is shown in Listing 1.

Use the Basic program shown in Listing 2 to enter and then to enable the machine code program shown in Listing 1.

After you have entered Listing 2, CSAVE it before RUNning it. You should see an '*' in the top right-hand corner of the screen. Try to erase this by any means you like and you will find the best you can do is to erase it momentarily (in fact a maximum of approximately 20 ms, the time taken between successive interrupts). The only way to erase the '*' is to disable the interrupt itself, or to disable the machine code program by:

POKE 30845,201

which POKEs a RET instruction (0C9 HEX) back into location 787D HEX.

Real-time system pointer display

When programming in Basic a useful feature would be to see a constantly updated display of various system pointers (eg start of program, end of program, start of free space etc) to aid in keeping track of the progress of these parameters.

To illustrate this principle simply, we will display the contents of the output latch. A copy of the latch contents is maintained at location 783B HEX (307779 decimal). The latch controls the following:

BIT	FUNCTION	0	1
0	speaker O/P #1	see note below	
1	unused		
2	cassette O/P	toggles according to data O/P	
3	mode control	Mode 0	Mode 1
4	background colour	green	buff
5	speaker O/P #2	see note below	
6	unused		
7	unused		

Note: During a key press 'beep' or execution of the SOUND command, the software toggles bit 0 and bit 5. When it does this, it first looks at the state of each bit and then inverts that state. Normally each bit (0 and 5) are the complement of each other, and the inversion of both at the same time gives a 'push-pull' like drive signal to the speaker. However, if both bits were the same, there would be no differential change when they are inverted, and so no output. You can therefore disable the 'beep' and the SOUND command by looking at both bits and then POKEing a value into location 783B HEX (30779 decimal) which makes them equal. That is, if the contents of 783B HEX are even, then POKE back into 783B HEX a value equal to (contents + 1). Conversely, if the contents are odd, POKE back a value of (contents − 1).

Getting back to the latch display — to indicate the state of each bit, we will display a '0' or '1' for each bit in the top right-hand corner of the screen.

The machine language source code is shown in Listing 3.

The Basic program in Listing 4 will enter and enable the machine code program of Listing 3. Note that Listing 4 is similar to Listing 2, so if you have already entered Listing 2 you can modify it to Listing 4. Once again, enter the Basic program (Listing 4), and CSAVE it before RUNning it. You should see the contents of the output latch displayed in binary in the top right-hand corner of the screen, reading from left to right, starting with bit 7 across to bit 0. Change the background colour (COLOR,0 and COLOR,1) and note the change in bit 4 in the display.

Cursor position pointer

Edit line number 900 to:

900 DATA 245,197,229,58,166,120,6

ReRUN the program.

This will display the horizontal cursor position pointer (0-31) from location 78A6 HEX (30886 decimal). Use the left/right cursor position arrows to move the cursor and observe the display.

Basic program pointers

Now edit line number 900 to:

900 DATA 245,197,229,58,249,120,6

ReRUN the program again.

This will display the LSB (Least Significant Byte) of the 'end of Basic program' pointer. Try adding extra lines to the Basic program and note the change in the display. For example, add the line:

1500 REM TEST

Note down the binary value displayed and then edit line 1500 to:

1500 TEST

Compare the new display value with the previous value.

This exercise reveals that although the short form remark symbol (') occupies two *screen* spaces *less* than the long form REM command, it needs two *more program memory* spaces to store it than the long form!

What next?

These given examples are very simple ones designed to illustrate the basic principle of using the video interrupt and do not show the full potential of the technique. I have written two programs which utilize this technique in a more complex fashion. The first of these is a real-time clock which is controlled by the internal clock of the VZ200. This gives a digital readout display in the upper right-hand corner of the screen. The real-time clock is implemented entirely in software (no need for extra hardware or modifications).

The second program demonstrates a split-screen graphics mode with one part of the screen having text and lo-res graphics, with the remainder in hi-res graphics.

Other applications

These are but a few of the many possible uses of the video interrupt. Other applications include:

- arcade games — synchronizing movement with the video raster rate to give smooth action. Mixed hi-res graphics and text for scoring, simulating instrumentation etc;
- stopwatch — event timer or lap-scorer;
- frequency counter — using the internal VZ200 clock to give the timing gate period; and
- real-time control — using the VZ200 as a component in a control system, eg burglar alarm.

The list could go on, as anything which requires a reasonably accurate time-keeping function or synchronization with the video display, is a possible candidate. Which all goes to show that it's not always rude to interrupt! ●